VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DATABÁZOVÁ VRSTVA INFORMAČNÍHO SYSTÉMU S APLIKAČNÍM ROZHRANÍM

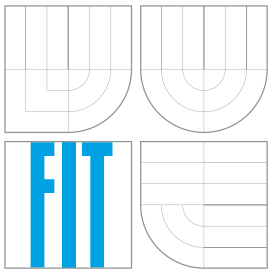
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB ŠPAČEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DATABÁZOVÁ VRSTVA INFORMAČNÍHO SYSTÉMU S APLIKAČNÍM ROZHRANÍM

INFORMATION SYSTEM DATABASE LAYER WITH AN APPLICATION INTERFACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB ŠPAČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2015

Abstrakt

Tato práce je součást studentského projektu nazvaného Yumified. Jedná se o mobilní aplikaci umožňující uživatelům hodnotit pokrmy ve stravovacích zařízeních. Uživatelé si dále mohou všechna hodnocení prohlížet. V této práci jsou popsány návrh a implementace aplikačního rozhraní a databáze sloužících ke sběru, uchování a poskytování informací. K tomu je využita architekturu aplikačního rozhraní REST, PHP rámec Symfony 2 a knihovna Doctrine pro práci s databází na vysoké úrovni.

Abstract

This thesis is a part of a student project called Yumified. It is a mobile application that allows users to rate melas in catering facilities. The users can also browse all the ratings. The thesis describes the design and implementation of an application interface and a database that serve to collect, store and provide the related information. For this purpose, we use the REST application interface, Symfony PHP framework and the Doctrine library for high-level database access.

Klíčová slova

databáze, aplikační rozhraní, framework, PHP, objektově relační mapování, MySQL, SOAP, REST, JSON, Symfony, Doctrine

Keywords

database,application interface, framework, PHP, object-relational mapping, MySQL, SOAP, REST, JSON, Symfony, Doctrine

Citace

Jakub Špaček: Databázová vrstva informačního systému s aplikačním rozhraním, bakalářská práce, Brno, FIT VUT v Brně, 2015

Databázová vrstva informačního systému s aplikačním rozhraním

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Špaček
12. května 2015

Poděkování

Děkuji Ing. Radku Burgetovi, Ph.D. za odbornou pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Využití chytrých telefonů stoupá přímo úměrně se zvyšujícím se počtem mobilních aplikací. Projekt Yumified, jehož součástí je tato práce, se zaměřuje na hodnocení pokrmů ve stravovacích zařízeních. Jedná se o mobilní aplikaci, kde si uživatelé mohou prohlížet již přidaná hodnocení a zároveň přidávat svá vlastní. Záměrem je přitom usnadnit uživateli výběr pokrmu v daném stravovacím zařízení nebo ve svém blízkém okolí, případně motivovat provozovatele stravovacích zařízení, aby zlepšili své služby.

Cílem této práce je vytvořit databázi a webové aplikační rozhraní. Databáze uchovává informace o navštívených stravovacích zařízeních, pokrmech, hodnocení pokrmů, uživatelích a dalších záznamech typu fotografie, vývoj kvality stravovacího zařízení a vztahy mezi uživateli. Informace jsou do databáze získávány prostřednictvím webového aplikačního rozhraní. To může komunikovat jak s aplikací Yumified, tak s dalšími aplikačními rozhraními jiných služeb, např. Foursquare. Skrze vlastní aplikační rozhraní lze také uložené informace třídit a poskytovat. Některé informace lze poskytnout komukoliv, např. výběr nejlépe hodnocených jídel v blízkém okolí, jiné jsou zase poskytovány pouze oprávněným uživatelům, např. body sesbírané za hodnocení pokrmů.

Mobilní aplikace Yumified, pro níž je vytvořena tato databáze, je vyvíjena pro operační systémy Android a iOS v rámci bakalářských prací studentů Jana Tomeška a Petra Bobáka.

Kapitola 2

Dostupné prostředky pro práci s databází na platformě PHP

Tato kapitola se věnuje dostupným prostředkům, které lze využít k implementaci aplikačního rozhraní na platformě PHP, komunikujícího s databází. Jsou zde posuzovány přístupy pro práci s databází na různých úrovních, architektury aplikačního rozhraní a dostupné PHP rámce.

2.1 Databáze a PHP

Tato sekce pojednává o přístupech do databáze na různých úrovních abstrakce. Uvádí příklady jednotlivých přístupů a srovnává jejich klady a zápory.

2.1.1 Standardní prostředky PHP

Práce s databází pomocí čistého PHP je možná, ale jedná se o naprostý základ. Samotné PHP implementuje množství funkcí pro připojení databáze a dotazování. Není proto potřeba žádných knihoven třetích stran, což může být výhoda pro uživatele začátečníky. Od verze 5.1.0. je v PHP zabudováno tzv. PDO[1], umožňující jednotné připojení a práci s různými typy databází na objektové úrovni.[1] Připojení k databázi a dotazování je popsáno následujícím příkladem:

```
// SQL - pripojeni k databazi
$dbhandle = mysql_connect("localhost", "user_name", "user_password")
            or die("Unable to connect to MySQL");

// SQL - dotazovani
$dotazSQL = $em->createNativeQuery('
                    SELECT type
                    FROM Auto a
                    WHERE a.Passengers = 5');
```

Přestože je pomocí PDO odstíněna různorodost databází, programátor stále může narazit na problémy, např. se zabezpečením proti útokům typu SQL injection. Při použití čistého

[1] PHP Data Objects

PHP je také programátor nucen pokaždé znova řešit tytéž problémy s ošetřením neplatných vstupních informací, datových typů a jiných běžných událostí, způsobujících poruchu nebo nekonzistenci dat. Proto se od tohoto přístupu upouští a je nahrazován novými postupy, které jsou řešeny v následujících kapitolách.

2.1.2 Query builder

Query Builder je zvláštní třída, která umožňuje vytvořit databázový dotaz postupným voláním jejích metod.[2] Tímto způsobem je aplikace chráněna proti útokům typu SQL injection, protože není nutné escapovat rizikové znaky. Query Builder bývá součástí tzv. PHP rámců, dále zvaných jako frameworky, což jsou sady knihoven řešící často vyskytující se problémy. Frameworkům bude věnována samostatná kapitola. PHP frameworky, které nemají vlastní query builder, většinou podporují jejich přidání v podobě externích knihoven. Takovými knihovnami jsou především Doctrine nebo Propel a přinášejí sebou další nástroje pro bezpečnější práci s databází.

```
// query builder - dotazovani
$dotazQB = $em->createQueryBuilder()
              ->select('type')
              ->from('Auto', 'a')
              ->where('a.Passengers = 5')->getQuery();
```

Nezávislým prostředkem, disponujícím query builderem, je knihovna NotORM. Klade důraz na jednoduchý přístup k datům a na minimální počet přenášených dat. Je použitelná pro většinu známých druhů databází díky základům postaveným na PHP PDO. Jak už název napovídá, nevyužívá objektově relačního mapování, které bude popsáno v následující kapitole. NotORM ale nepřináší řešení dříve popsaných problémů s kontrolou korektnosti vstupních dat. Proto je nutné pro velké projekty uvažovat její využití v kombinaci s dalším frameworkem.[3][4]

2.1.3 Objektově relační mapování

Objektově relační mapování, zkratkou ORM, je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.[5] Jeden záznam tabulky databáze je tedy reprezentován objektem, který má své metody pro práci se svými atributy. Jde o moderní způsob manipulace s daty, nesoucí spoustu výhod oproti klasickému použití čistých SQL příkazů. Mezi frameworky a knihovnami, umožňujícími ORM, vynikají především Doctrine a Propel.

Mezi výhody patří přítomnost abstraktní databázové vrstvy, která poskytuje jednotný zápis SQL příkazů. Možnost zapouzdřit každý záznam tabulky do samostatného objektu přináší lepší manipulaci a orientaci v kódu. Testování lze snadno provádět vestavěnými unit testy, které dokáží testovat i rozšířené třídy. Frameworky, podporující ORM, dokáží na základě definice databázových objektů vytvořit uživatelské rozhraní pro základní operace, tzv. CRUD[2].[6]

Nevýhodou je vyšší paměťová a procesorová náročnost. Při práci s větším počtem záznamů, kdy pro každý záznam tabulky vzniká objekt, lze rychle narazit na paměťový limit. Proto je vhodné pečlivě vážit, se kterými záznamy je aktuálně potřeba pracovat a je tedy nutné je načíst do paměti.[6]

[2] Create, Read, Update, Delete

Příklad práce s databázovými objekty:

```
// vytvoren novy objekt, reprezentujici zaznam do tabulky Auto
$auto = new Auto();
$auto->setType('Fabia');     // prirazeni atributu
$auto->setColor('red');
$auto->setPassengers(5);
$auto->save();               // ulozeni do databaze
// ziskani primarniho klice zaznamu a jeho vypsani
echo $auto->getId . " added to DB\n";
```

2.1.4 DQL

Již zmíněný Doctrine poskytuje kromě query builderu také dotazovací jazyk DQL[3]. Jeho syntaxe je velice podobná běžnému SQL, je však založeno na jiných principech a leží v objektové vrstvě aplikace. Běžný SQL dotaz pracuje s databázovými tabulkami a jejich atributy, DQL používá jména Doctrine entit a jejich členských proměnných. Výsledkem SQL dotazu je vždy tabulka, výsledkem DQL dotazu jsou načtené instance entit.[7] Při správném použití DQL je riziko SQL injection minimalizováno. Nevýhodou DQL je nemožnost vytvářet a vkládat nové záznamy do tabulky pomocí příkazu INSERT. Kvůli tomu, že DQL leží na objektové vrstvě, nikoliv na databázové, by vytvoření nového záznamu způsobilo nekonzistenci entit uvnitř Doctrine. Toto je velmi důležité omezení, protože vyvíjený produkt má především ukládat informace do databáze.

```
// DQL - dotazovani
$dotazDQL = $em->createQuery('
                    SELECT type
                    FROM Auto a
                    WHERE a.Passengers = 5');
```

2.2 Webové aplikační rozhraní

Webové aplikační rozhraní, nazývané také jako webová služba, zkratkou API, je softwarový systém umožňující interakci dvou strojů na síti.[8]

Mezi významnější technologie umožňující tvorbu webového aplikačního rozhraní patří protokoly: XML-RPC, SOAP, REST, CORBA, GIOP a DCOM. Ze zmíněných protokolů lze pro použití v tomto projektu uvažovat pouze XML-RPC, SOAP a REST. Ostatní protokoly nejsou příliš používané, nejspíše pro svůj zápis zpráv pro komunikaci v binární podobě, a nesou s sebou určitá omezení, např. DCOM je vyvinut společností Microsoft a je úzce spjat s její infrastrukturou.[9][10]

V dalším kroku je vhodné vyloučit z výběru i protokol XML-RPC, neboť jeho vývoj byl ukončen. Náhradou za něj se stal již zmiňovaný protokol SOAP, který vychází z jeho předlohy a dodržuje jeho konvence.[9]

Následující část popisuje zbylé dva protokoly SOAP a REST a komentuje jejich vhodnost pro použití pro tento projekt.

[3] Doctrine Query Language

2.2.1 SOAP

Simple Object Access Protocol, zkratkou SOAP, je protokolem pro výměnu zpráv přes síť, hlavně pomocí HTTP[4]. Formát SOAP tvoří základní vrstvu komunikace mezi webovými službami a poskytuje prostředí pro tvorbu složitější komunikace.[9]

Zasílané zprávy jsou většinou složeny ze dvou částí, Envelope a Body. SOAP sám o sobě tvoří pouze obálku pro data přenášená uvnitř části Body. Aby mohla být data přenášena uvnitř těla požadavku, je pro zasílání využívána HTTP metoda POST. Data jsou zapsána pomocí XML[5] a jsou předána službě ke zpracování. Pro jednotnou interpretaci dat existuje formát pro popis webové služby, zkratkou WSDL. Průběh celého procesu komunikace je popsán protokolem RPC.[11]

WSDL

Web Services Description Language, zkratkou WSDL, popisuje funkce nabízené webovou službou a způsob, jak s ní komunikovat a dotazovat se. Zapisuje se v XML formátu. Zpravidla tedy popisuje SOAP komunikaci. Podporované operace a zprávy jsou popsány abstraktně a potom se omezují na konkrétní síťový protokol a formát zprávy. WSDL tedy popisuje veřejné rozhraní webové služby.[12]

Typicky obsahuje elementy:

- `types` pro definici datových struktur používaných ve zprávách,
- `message` pro definici formátu předávaných zpráv pomocí dříve definovaných datových typů,
- `portType` pro abstraktní definici typů portů, které jsou službou podporovány,
- `binding` pro navázání určitého typu portu na konkrétní protokol a formát přenosu zpráv,
- `service` pro samotnou definici služby.[11]

RPC

Remote procedure call (RPC, vzdálené volání procedur) je technologie dovolující programu vykonat proceduru, která může být uložena na jiném místě než je umístěn sám volající program. Příkladem může být výpočet funkce na jiném počítači v síti.[13] Průběh je popsán následujícími kroky:

1. Zabalení parametrů a identifikátoru procedury do formy vhodné pro přenos,
2. odeslání balíčku,
3. přijetí balíčku na vzdáleném místě, rozbalení a určení požadované procedury,
4. provedení procedury,
5. zabalení výsledku procedury,
6. odeslání balíčku zpět,

[4] Hypertext Transfer Protocol
[5] Extensible Markup Language

7. příjem balíčku a rozbalení,

8. předání výsledku volající proceduře.[13]

Syntaxe SOAP zprávy

V hlavičce zprávy se nachází upřesnění příjemce, odesilatele, kódování a může se zde nacházet název operace, kterou má služba vykonat. Příklad hlavičky zprávy:

```
POST /prijemce/pozadavku.php HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8; action="operace"
```

V těle se nachází především parametry vyžadované volanou funkcí. Tělo dotazu je popsáno následujícím příkladem, který lze nalézt na straně 325 v knize "PHP a XML".[11]

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope">
  <SOAP-ENV:Body>
    <Udaje xmlns="http://example.com/BMI">
      <Jmeno>Pepa</Jmeno>
      <Vaha>80</Vaha>
      <Vyska>180</Vyska>
    </Udaje>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Webová služba po úspěšném zpracování požadavku vytvoří novou SOAP obálku a do těla zprávy vloží odpověď. Ta je znázorněna následujícím příkladem, který lze nalézt ve stejné knize na straně 326.[11]

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:response="http://example.com/BMI">
  <env:Body>
    <response:BMI>24.69</response:BMI>
  </env:Body>
</env:Envelope>
```

Není nutné, aby programátor znal přesnou strukturu zasílaných zpráv, protože je lze většinou generovat za pomoci PHP metod. Následující příklad, který byl převzat ze stejné knihy[11] s částečnými úpravami, demonstruje jak použít SOAP v programu.

```
// vytvoreni klienta pro praci s webovou sluzbou na zaklade WSDL
$client = new SoapClient("bmi-service.wsdl");

// vytvoreni tridy pro zapis parametru pozadavku
$params = new ParamsClass();
$params->Jmeno = "Pepa";
$params->Vaha = 85;
$params->Vyska = 180;

// zavolani webove sluzby a ulozeni vysedku
$result = $client->SpocitejBMI($params);
```

V proměnné `$result` se nyní nachází výsledek metody `SpocitejBMI`. Na první pohled přitom nelze rozeznat, že se nejedná o metodu lokálního objektu, ale že právě došlo k volání vzdáleného procesu přes síť. Veškerý XML obsah zprávy byl automaticky vytvořen, odeslán, přijat a přečten při volání metody `SpocitejBMI`.

2.2.2 REST

Representational state transfer, zkratkou REST, je architektura rozhraní, navržená pro distribuované prostředí, umožňující jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP metod.[14]

Na rozdíl od SOAP, který je orientován procedurálně, REST je orientován datově, což způsobuje naprosto odlišné chování. SOAP definuje vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k tzv. zdrojům[6]. Zdroje mohou být jak data, tak datově vyjádřené stavy procedur a každý zdroj je jednoznačně určen pomocí URI[7]. Přístup ke zdrojům je umožněn pouze pomocí operací CRUD (Create, Read/Retrieve, Update, Delete), jež odpovídají HTTP metodám POST, GET, PUT, DELETE v tomto pořadí. Na rozdíl od SOAP, který vyžaduje komunikaci výhradně ve formátu XML, zdroje pro REST mohou být reprezentovány libovolným způsobem. Nejčastěji používané formáty jsou JSON[8], XML, RSS a ATOM.[15][14][16]

Každá operace, jež má být provedena pomocí REST, musí být realizována jednou z CRUD metod. Protože je REST bezstavový, musí každý požadavek obsahovat veškeré potřebné informace k provedení operace. Pokud provedení operace vyžaduje autentizaci, měly by tyto údaje být v požadavku přiloženy, nejčastěji v zašifrované podobě. Je nutno pamatovat, že citlivé informace, jako jsou jméno a heslo, by se nikdy neměly předávat pomocí metody GET, ale vždy hlavně pomocí metody POST. Požadavek GET se ukládá do cache zařízení a také do historie prohlížeče, kde by byly tyto citlivé údaje snadno dostupné.[17][18]

Create

Create je realizováno HTTP metodou POST. Protože jde o vytváření nového zdroje, potřebná URI ještě neexistuje a je nutné provést Create např. pomocí nástroje CURL[9]. Tomu je potřeba zadat autentizační údaje, data a společný identifikátor, tzv. koncový bod. Následující příklad demonstruje vytvoření nového zdroje a byl převzat z webu[15].

```
//      autentizace      data
curl -u user:password -d "status=Zkousime REST API"
        http://twitter.com/statuses/update.xml
//      koncový bod
```

Read/Retrieve

Read/Retrieve je realizováno HTTP metodou GET. V době vytvoření požadavku zdroj existuje a je nutné znát jeho URI, nebo alespoň znát princip jak jej sestavit. Tyto informace většinou bývají volně poskytovány provozovatelem API. Do požadavku je možno přidat parametry pro úpravu či filtraci výsledku.[15]

[6] anglicky resources
[7] Uniform Resource Identifier
[8] JavaScript Object Notation
[9] Client URL Library

Následující příklad demonstruje načtení zdroje s úpravou výsledku pomocí parametrů a byl převzat z webu[19].

```
GET
https://api.twitter.com/1.1/statuses/user_timeline.json?
    screen_name=twitterapi&count=2
```

Výsledkem požadavku jsou dvě zprávy od uživatele Twitteru "twitterapi", získané ve formátu JSON. Parametry, upravující výsledek, jsou uvedeny v části za otazníkem '?' a jsou od sebe odděleny ampersandem '&'. Takovýchto parametrů může být v dotazu uvedeno i více, jejich zpracování pak provádí aplikační rozhraní.

Update

Update je realizováno HTTP metodou PUT. Tato metoda nemusí být vždy podporována, ale lze ji provést podobně jako Create. Je možné opět využít nástroj CURL, který vyžaduje autentizační údaje, nová (změněná) data a URI upravovaného zdroje. Následující příklad demonstruje upravení zdroje a byl převzat z webu[15].

```
//      autentizace      zmenena data
curl -u user:password -d "url=http://zdrojak.root.cz"
        http://twitter.com/account/update_profile.xml
//      uri zmeneneho zdroje
```

Delete

Delete je realizováno HTTP metodou DELETE. Ani tato metoda ale nemusí být vždy podporována a tak je většinou nahrazována metodou POST s parametrem určujícím provedení Delete, nebo pomocí speciální URI. Následující příklad demonstruje smazání zdroje ve dvou variantách a byl převzat z webu[15].

```
// DELETE je podporovano
DELETE /api/user/pepa
Host: www.server.cz

// DELETE neni podporovano, nahrada specialni URI
http://twitter.com/statuses/destroy/číslo.format
```

JSON

JavaScript Object Notation, zkratkou JSON, je způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích, strukturách nebo agregována v objektech.[20]

Architektura rozhraní REST umožňuje předávat data v téměř libovolném formátu. Pro předávání dat mezi databází a aplikací Yumified byl zvolen formát JSON. Je velmi rozšířený a data pomocí něj přenášejí i všechna další aplikační rozhraní, s nimiž má výsledný produkt komunikovat.

Výhodou u JSON je znatelně kratší zápis při zachování veškeré hierarchie a logiky. Následuje krátký příklad syntaxe JSON.

```
{
  "jmeno": "Eduard",
  "prijmeni": "Patička",
  "adresa": {
      "obec": "Brno",
      "cp": 42,
      "psc": 61200,
  },
}
```

2.2.3 Zvolená architektura aplikačního rozhraní

Na základě zjištěných informací byla pro použití v tomto projektu zvolena architektura REST, především pro svou univerzálnost a nezávislost na formátu předávaných dat. Obliba této architektury je na vzestupu, především u vývojářů populárních mobilních aplikací a sociálních sítí. Aplikační rozhraní s architekturou REST využívá Twitter, Google+ i Foursquare. Facebook již REST API nepoužívá, ale nahradil jej vlastním, tzv. Graph API, které je od REST API odvozeno.[21][22][23][24]

Výsledný produkt má komunikovat s rozhraním Foursquare, což by v případě volby SOAP znamenalo podporu obou architektur. Také vývojáři aplikace Yumified implementují REST API.

2.3 PHP framework

Framework (aplikační rámec) je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.[25]

Použití frameworku tak přináší spoustu výhod ve formě usnadnění a urychlení vývoje produktu. Vývojář se může zaměřit na podstatu produktu a nezdržovat se tak často a opakovaně vyskytujícími se problémy.

PHP frameworků neustále přibývá a ne všechny jsou vhodné pro práci s databází. Proto je nutné framework vybrat podle určitých kritérií. Sledované vlastnosti jsou:

- Podpora ORM,
- přítomnost query builderu, nebo jiných způsobů dotazování minimalizujících riziko SQL injection,
- vhodnost pro střední a velké projekty, nejlépe upřednostňující MVC,
- kvalitní dokumentace a aktivní komunita vývojářů.

2.3.1 Model–View–Controller

Model–view–controller (MVC) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní.[26] Jde o moderní přístup k vývoji aplikací, vyžaduje modularitu a zpřehledňuje kód. Umožňuje současnou práci programátora a designéra na tomtéž projektu, aniž by docházelo k vzájemnému narušování kódu.

2.3.2 Dostupné PHP frameworky

Při výběru MVC frameworku je výhodné zaměřit se na ty nejznámější a nejpoužívanější, u nichž lze očekávat kvalitní dokumentaci a velkou aktivní komunitu vývojářů. Na základě žebříčku na webu SitePoint[27] byly vybrány a posuzovány následující frameworky. Popularita a aktivní komunita vývojářů byly určeny pomocí množství řešených témat pro každý framework na webu Stack Overflow.

Laravel

Výhodami tohoto frameworku jsou modularita a snadná rozšiřitelnost pomocí balíků. Zabudovaná podpora ORM a query builderu. Dokumentace je na velice vysoké úrovni a na webu Stack Overflow[10] lze najít více než 15 000 otázek a odpovědí.[28]

Phalcon

Tento framework je primárně zaměřen na rychlost. Výsledný program není interpretovaný, jak je u PHP běžné, ale kompilovaný, což přináší výrazné urychlení. Aby bylo možné skript přeložit, framework je implementován v kombinaci jazyků C/C++ a PHP v podobě tzv. C rozšíření. To znamená rozkouskování PHP kódu a jeho, většinou automatizovaný, přepis do jazyka C/C++.[29] Nativně podporuje ORM, má vlastní query builder a snadno lze přidat knihovny třetích stran. Dokumentace je kvalitní, ale na webu Stack Overflow[11] je k nalezení jen asi 700 otázek a odpovědí k tomuto frameworku.[30]

Symfony 2

Symfony 2 je framework pro střední a velké projekty. Je velmi robustní a nabízí spoustu nástrojů již v základu. Další rozšíření lze snadno přidávat ve formě balíků.[31] Podporuje ORM, query builder i DQL, protože je úzce spjat s knihovnou Doctrine.[32] Přesto vývojáři ponechává volnost vybrat si i z jiných nástrojů pro práci s databází, např. Propel. Dokumentace je velmi rozsáhlá a detailní a na webu Stack Overflow[12] lze nalézt více než 25 000 otázek a odpovědí.

CodeIgniter

CodeIgniter je framework pro středně velké projekty. Obsahuje jen několik málo knihoven a proto je velmi rychlý a snadný k naučení. Lze jej rozšiřovat přidáváním jak oficiálních knihoven, tak i knihoven třetích stran. Knihovnu ORM je možné přidat pomocí jednoho z mnoha externích balíků speciálně pro něj, např. datamapper[13]. Dokumentace je na velmi dobré úrovni a lze nalézt více než 33 000 zodpovězených dotazů[14].[33]

Yii

Framework Yii byl vyvíjen s ohledem na použití pro firmy, lze jej však využít i pro osobní menší projekty. Obsahuje spoustu knihoven, především pak podporuje ORM a query builder

[10] http://stackoverflow.com/questions/tagged/laravel
[11] http://stackoverflow.com/questions/tagged/phalcon
[12] http://stackoverflow.com/questions/tagged/symfony2
[13] http://datamapper.wanwizard.eu/
[14] http://stackoverflow.com/questions/tagged/codeigniter

a lze jej dále rozšiřovat. Dokumentace je na dobré úrovni a lze nalézt více než 12 000 dotazů a odpovědí[15].[34]

CakePHP

CakePHP je jeden ze starších frameworků a také jeden z interpretačně nejpomalejších. Je navržen tak, aby práce s ním byla snadná a rychlá. Již v základu podporuje ORM a query builder. Hlavní nevýhodou zůstává jeho pomalost. Framework obsahuje spoustu řádků kódu, třebaže dobře komentovaného.[28] Dokumentace je na dobré úrovni a lze nalézt více než 21 000 řešených problémů[16].

Zend Framework 2

Jde o jeden z nejznámějších a nejpoužívanějších PHP frameworků. Je velmi robustní, primárně zaměřený na velké projekty a na použití ve firmách. Podpora ORM je zajištěna snadnou rozšiřitelností o knihovnu Doctrine.[35] Dokumentace je obsáhlá, ale ne příliš vhodná pro začátečníky. Framework sebou přináší vlastní IDE pro usnadnění vývoje produktů, které vývojář může, ale nemusí použít.[36][37] Komunita okolo Zendu je velmi rozsáhlá, přesto lze na webu Stack Overflow nalézt jen necelých 6 000 řešených problémů[17]. V případě hledání odpovědi existuje spousta dalších webů, některých přímo zaměřených na tento framework.

2.3.3 Zvolený PHP framework

Na základě sesbíraných informací a provedení několika jednoduchých demonstračních cvičení byl pro realizaci tohoto projektu zvolen framework Symfony 2 verze 2.6. Především pro svou propojenost s Doctrine, kvalitní a rozsáhlou dokumentaci a možnost rozšíření o knihovny pro implementaci a práci s REST aplikačním rozhraním, zvanou FOSRestBundle[18]. Rychlého vyřizování požadavků je docíleno implementací vlastní chytré cache. Yumified bude do budoucna vyžadovat kvalitní framework pro střední a velké projekty, čemuž nejlépe odpovídá Symfony 2.

[15] http://stackoverflow.com/questions/tagged/yii
[16] http://stackoverflow.com/questions/tagged/cakephp
[17] http://stackoverflow.com/questions/tagged/zend-framework2
[18] https://github.com/FriendsOfSymfony/FOSRestBundle

Kapitola 3

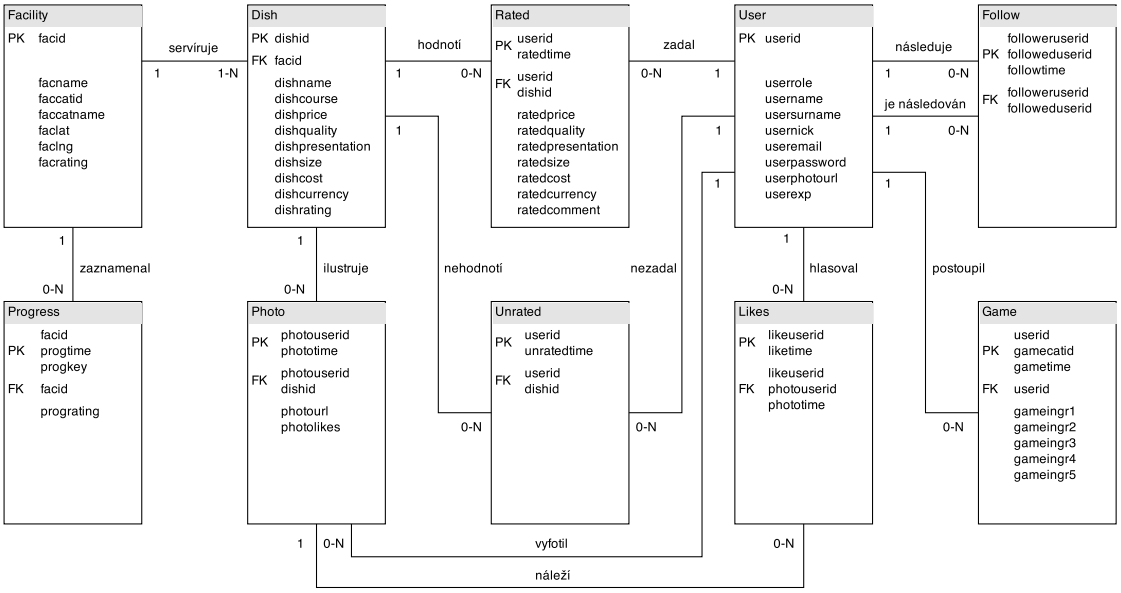
Návrh

Aplikace Yumified primárně vyžaduje informace o stravovacích zařízeních, pokrmech, hodnocení pokrmů a uživatelích. Pro větší potěšení z používání aplikace jsou přidány herní prvky na způsob sbírání odměn za přínos nových informací. Yumified chce uživatelům dát možnost vytvářet komunity stejným způsobem jako Twitter, tedy pomocí následování jednoho uživatele jiným uživatelem. Dále nabízí uživateli možnost pořídit fotografie hodnoceného pokrmu. Pořízených fotografií může být více a vždy se jako úvodní fotografie pokrmu zobrazí ta, která obdrží od uživatelů nejlepší hodnocení. Za takovéto fotografie získává jejich autor další odměny. Pokud si uživatel přeje ohodnotit pokrm později, je nutné tuto informaci uchovat. Posledním prvkem je průběžné uchovávání vývoje kvality stravovacího zařízení. Pro lepší pochopení způsobu použití jsou popsány tři z mnoha možných scénářů, z čehož první je převažující a nejobsáhleji popisující:

- Uživatel má chuť na zaručeně dobré jídlo. V aplikaci si zobrazí kategorii nejlépe hodnocených pokrmů pro své blízké okolí, nazvanou "Top around". Pro každý pokrm si může zobrazit detailní informace a všechna hodnocení. Pro každý pokrm si také může nechat zobrazit informace o zařízení, kde jej servírují. Pokud se rozhodne podnik navštívit a pokrm si objednat, může jej ohodnotit, případně vyfotit. Za každé hodnocení pokrmu, přidání nového pokrmu a dalších příspěvků do databáze získává uživatel body a zkušenosti, které slouží jako ukazatele aktivity a prestiže.
- Uživatel A, kterého uživatel B následuje, nedávno ohodnotil některý pokrm jako velmi dobrý. Uživatel B se rozhodne pokrm vyzkoušet také a vytvořit si vlastní názor.
- Uživatel přijde do jediné restaurace v okolí, a neví co dobrého si zde objednat. V aplikaci nalezne danou restauraci a nechá si zobrazit pokrmy. Buď vybere jeden z již hodnocených, nebo se rozhodne vyzkoušet nějaký nový. Může tak do databáze přidat nový pokrm, vzápětí nové hodnocení a získat tak body a zkušenosti.

3.1 Návrh databáze

Následuje popis jednotlivých entit, jejich atributů a vazeb. Entity jsou rozvrženy s ohledem na možnou rozšiřitelnost. Pro ilustraci byl sestaven návrh schématu databáze na obrázku 3.1.



Obrázek 3.1: Návrh schématu databáze.

3.1.1 Facility

Entita `Facility` reprezentuje stravovací zařízení. Databáze uchovává pro každé zařízení jediný záznam, který je vytvořen až tehdy, kdy je pro něj vložen první pokrm. Informace o zařízení jsou získány z Foursquare, prostřednictvím jeho API.

Atributy:

- `facid` — primární klíč stravovacího zařízení, není vytvářen ale je přebírán z Foursquare, kde je označen jako VENUE_ID,
- `facname` — název zařízení,
- `faccatid` — id kategorie zařízení,
- `faccatname` — název kategorie zařízení, např. rychlé občerstvení,
- `faclat` — zeměpisná šířka stravovacího zařízení,
- `faclng` — zeměpisná délka stravovacího zařízení,
- `facrating` — celkové hodnocení zařízení, vypočtené z celkového hodnocení všech jeho pokrmů.

Pro každé stravovací zařízení v databázi je ukládán záznam jeho aktuálního celkového hodnocení. Jedno stravovací zařízení může být svázáno s 0–N záznamy vývoje celkového hodnocení (`Progress`) a s 1–N pokrmy (`Dish`).

3.1.2 Progress

Účelem této entity je zaznamenávat aktuální celkové hodnocení stravovacího zařízení po každé jeho změně. Nashromážděná data jsou určena k vykreslení grafu vývoje hodnocení stravovacího zařízení.

Atributy:

- `facid` – součást složeného primárního klíče, jedná se o cizí klíč z tabulky `Facility`,
- `progtime` – součást složeného primárního klíče, jedná se o čas vytvoření záznamu,
- `progkey` – součást složeného primárního klíče, jedná se o náhodné číslo pro zajištění jednoznačné identifikace záznamu v databázi,
- `prograting` – uložená hodnota celkového hodnocení stravovacího zařízení.

Jeden záznam hodnocení může být svázán s právě jedním zařízením (`Facility`).

3.1.3 Dish

Entita `Dish` reprezentuje jeden pokrm. Záznam o daném pokrmu nezávisí na hodnocení, které může, ale nemusí být vloženo při jeho přidání. V atributech jsou uchovávány průměrné hodnoty různých kategorií hodnocení, které se aktualizují při vložení nového hodnocení daného jídla, aby se tyto hodnoty nemusely počítat při každém prohlížení uživatelem.

Atributy:

- `dishid` – primární klíč záznamu, vytvořen pomocí `facid` a auto–increment,
- `facid` – cizí klíč zařízení z tabulky `Facility`,
- `dishname` – název pokrmu,
- `dishcourse` – informace o chodu nebo druhu pokrmu (hlavní chod, předkrm, atd.),
- `dishprice` – průměrné hodnocení poměru kvality pokrmu ku jeho ceně,
- `dishquality` – průměrné hodnocení kvality pokrmu,
- `dishpresentation` – průměrné hodnocení vizuální prezentace,
- `dishsize` – průměrné hodnocení velikosti pokrmu,
- `dishcost` – cena pokrmu,
- `dishcurrency` – měna, ve které je zadána cena pokrmu,
- `dishrating` – hodnota vzniklá zprůměrováním všech průměrných hodnocení.

Jeden pokrm může být svázán s právě jedním stravovacím zařízením (`Facility`), s 0–N fotografiemi (`Photo`), s 0–N hodnoceními (`Rated`) a s 0–N záznamy o odloženém hodnocení (`Unrated`).

3.1.4 Photo

Entita reprezentuje fotografii pokrmu. Pokrm nemusí mít fotografii, stejně tak jich může mít nepočítaně. Fotografie je možné hodnotit. Nejlépe hodnocená fotografie se u pokrmu zobrazuje jako primární.

Atributy:

- `photouserid` – součást složeného primárního klíče, jedná se o cizí klíč z tabulky `User`,
- `phototime` – součást složeného primárního klíče, jedná se o čas přidání fotografie,
- `dishid` – cizí klíč z tabulky `Dish`, určující pokrm na fotografii,
- `photourl` – URL odkazující na fotografii,
- `photolike` – počet hodnocení fotografie.

Jedna fotografie může být svázána s právě jedním pokrmem (`Dish`), mohla být pořízena právě jedním uživatelem (`User`) a mohla získat 0–N hodnocení (`Likes`).

3.1.5 Rated

Entita `Rated` reprezentuje jedno hodnocení konkrétního pokrmu.

Atributy:

- `userid` – součást složeného primárního klíče, jedná se o cizí klíč z tabulky `User`,
- `ratedtime` – součást složeného primárního klíče, jedná se o čas přidání hodnocení,
- `dishid` – cizí klíč z tabulky `Dish`, určující hodnocené jídlo,
- `ratedprice` – hodnocení poměru kvality pokrmu ku jeho ceně,
- `ratedquality` – hodnocení chuti pokrmu,
- `ratedpresentation` – hodnocení vizuální prezentace pokrmu,
- `ratedsize` – hodnocení velikosti pokrmu,
- `ratedcost` – cena pokrmu,
- `ratedcurrency` – měna uvedené ceny pokrmu,
- `ratedcomment` – celkové hodnocení pokrmu, vzniklé zprůměrováním všech hodnocení.

Jedno hodnocení mohlo být zadáno právě jedním uživatelem (`User`) pro právě jeden pokrm (`Dish`).

3.1.6 User

Entita `User` reprezentuje účet uživatele.

Atributy:

- `userid` – primární klíč záznamu, vytvořen pomocí auto–increment,
- `userrole` – role uživatele v systému,
- `username` – křestní jméno uživatele,
- `usersurname` – příjmení uživatele,
- `usernick` – přezdívka,
- `useremail` – email uživatele,
- `userpassword` – přihlašovací heslo uživatele,
- `userphotourl` – odkaz na profilovou fotografii uživatele,
- `userexp` – množství nasbíraných zkušeností.

Jeden uživatel může zadat 0–N hodnocení (`Rated`), vytvořit 0–N záznamů o odloženém hodnocení (`Unrated`), pořídit 0–N fotografií pokrmů (`Photo`), ohodnotit 0–N fotografií pokrmů (`Likes`), následovat 0–N uživatelů (`Follow`), být následován 0–N uživateli (`Follow`) a sbírat body v 0–N herních kategoriích (`Game`).

3.1.7 Game

Entita `Game` reprezentuje záznam herní kategorie. Uchovává počty nasbíraných odměn za ohodnocení pokrmů. Záznam vzniká zároveň s vytvořením hodnocení pokrmu v nové kategorii stravovacího zařízení (sushi bar, pizzerie, fast food, atd.).

Atributy:

- `userid` – součást složeného primárního klíče, cizí klíč z tabulky `User`,
- `gamecatid` – součást složeného primárního klíče, id kategorie stravovacího zařízení,
- `gametime` – čas založení hry dané kategorie,
- `gameingr1-5` – počet nasbíraných bodů různých kategorií.

Jedna herní kategorie je svázána s právě jedním uživatelem (`User`).

3.1.8 Unrated

Entita `Unrated` uchovává záznamy o odloženém hodnocení pokrmů. Pro jedno jídlo a jednoho uživatele může existovat vždy nejvýše jeden záznam o odloženém hodnocení.

Atributy:

- `userid` – součást složeného primárního klíče, cizí klíč z tabulky `User`,
- `unratedtime` – součást složeného primárního klíče, čas vložení záznamu o odloženém hodnocení,

- `dishid` – cizí klíč z tabulky `Dish`, určující neohodnocené jídlo.

Jedno odložené hodnocení je svázáno s právě jedním jídlem (`Dish`) a s právě jedním uživatelem (`User`).

3.1.9 Likes

Entita `Likes` uchovává záznamy o hodnocení fotografií pokrmů.
Atributy:

- `likeuserid` – součást složeného primárního klíče, cizí klíč z tabulky `User`,
- `liketime` – součást složeného primárního klíče, čas vložení hodnocení,
- `photouserid` – cizí klíč z tabulky `User`, určující uživatele, který pořídil hodnocenou fotografii,
- `phototime` – čas vytvoření záznamu fotografie pro jednoznačné určení hodnocené fotografie.

Jeden záznam o hodnocení je svázán s právě jednou fotografií (`Photo`) a s právě jedním hodnotícím uživatelem (`User`).

3.1.10 Follows

Entita `Follows` uchovává záznamy o vztazích mezi uživateli.
Atributy:

- `followeruserid` – součást složeného primárního klíče, cizí klíč z tabulky `User` určující uživatele který následuje jiného uživatele,
- `followeduserid` – součást složeného primárního klíče, cizí klíč z tabulky `User` určující uživatele který je následován jiným uživatelem,
- `followtime` – součást složeného primárního klíče, čas vytvoření záznamu o následování.

Jeden záznam o následování svazuje vždy dva různé uživatele (`User`). Tato provázanost může být vždy jedním či druhým směrem, nikdy však pro jeden záznam oběma směry.

3.2 Návrh aplikačního rozhraní

Aplikační rozhraní musí dokázat poskytnout minimálně dva druhy URI. První druh ukládá data do databáze, druhý druh získává data z databáze. Dodatečnými druhy jsou editace záznamů v databázi, což přichází v úvahu pouze u účtu uživatele, a smazání záznamu, což momentálně není řešeno a tato možnost by byla přidělena pouze administrátorovi databáze. Rozhraní je vyvíjeno primárně za účelem využití v kombinaci s mobilní aplikací Yumified, proto jsou následující kapitoly zaměřeny na nejdůležitější body poskytování informací pro potřeby této aplikace.

3.2.1 Seznam stravovacích zařízení

Vyvíjená databáze ukládá stravovací zařízení až ve chvíli, kdy je pro něj uložen první pokrm. To znamená, že databáze obsahuje jen nejnutnější minimum záznamů o zařízeních, což pro uspokojení uživatelů nestačí. Proto je v tomto bodě vhodné využít databázi Foursquare, která obsahuje obrovské množství míst, památek, atrakcí a v tomto případě hlavně také stravovacích zařízení. K jejich získání stačí zaslat správný dotaz s parametry zeměpisné šířky a délky. Přesnějších výsledků lze dosáhnout specifikací dalších volitelných parametrů, které budou detailněji popsány v části implementace.

Získaná odpověď pak obsahuje seznam blízkých zařízení i s jejich detailním popisem. Tyto informace je nutné nejprve filtrovat a seskládat do odpovědi takového tvaru, jaký požaduje aplikace. K těm zařízením, které již ve vyvíjené databázi mají záznamy, lze navíc připojit vlastní nasbírané informace, například o kvalitě zařízení. Teprve poté je možné ji odeslat uživateli.

3.2.2 Detail stravovacího zařízení

V tomto bodě se opět naráží na problém minimálního počtu zařízení ve vlastní databázi. Ta navíc uchovává jen ty nejnutnější informace, takže například otevírací dobu, adresu, fotografie apod. je potřeba získat opět z databáze Foursquare. Toho je dosaženo opět jediným dotazem na Foursquare API, tentokrát s jediným parametrem, identifikátorem stravovacího zařízení.

Získaná odpověď opět obsahuje obrovskou spoustu informací, které je nutno přebrat, sestavit do odpovědi pro aplikaci a případně doplnit některé informace navíc z vlastní databáze.

3.2.3 Seznam hodnocených pokrmů ve stravovacím zařízení

Pro zisk tohoto seznamu již není nutné využívat Foursquare a plně postačují data z vlastní databáze. Do požadavku je potřeba obsáhnout pouze parametr identifikátoru stravovacího zařízení. Z vlastní databáze jsou získány všechny záznamy o pokrmech s cizím klíčem odpovídajícím zadanému identifikátoru zařízení.

Získané záznamy je vhodné dále filtrovat podle volitelných kritérií, například pokud chce uživatel pouze deset nejlépe hodnocených pokrmů. Z vybraných výsledků je potřeba sestavit odpověď v očekávaném tvaru a teprve tu odeslat.

3.2.4 Detail pokrmu

Základní informace o pokrmu jsou pouze název a chod. Další položky jako cena, kvalita a chuť jsou přidány, až pokud bylo k jídlu vloženo hodnocení. Další nepovinnou položkou je fotografie. Při vytváření detailu pokrmu je tedy vhodné získat záznam nejen z tabulky pokrmu, ale také z tabulek hodnocení a fotografií. Pokud však tyto dodatečné informace v databázi nejsou, výsledek je odeslán bez nich a uživatel tak může být motivován k jejich přidání, za což bude odměněn body a zkušenostmi.

3.2.5 Seznam nejlépe hodnocených pokrmů v blízkém okolí

Zisk takovéhoto výsledku je kombinací předchozích zmíněných postupů. Tyto informace jsou veřejně dostupné, není tedy potřeba aby byl uživatel přihlášen. Je ale nutné zadat parametry zeměpisné šířky a délky. Ty jsou následně přeposlány jako parametry dotazu na Foursquare

API. Odpovědí je seznam blízkých stravovacích zařízení, pro které jsou ve vlastní databázi hledány nejlépe hodnocené pokrmy. Odpověď pro uživatele je poté složena pouze ze seznamu pokrmů, kde je navíc ke každému připojen identifikátor zařízení, které jej servíruje.

3.2.6 Registrace uživatele

Aby mohl uživatel přidávat pokrmy, hodnocení a fotky, musí být nejprve registrován. Registraci lze provést jediným POST dotazem, kde zašle údaje o sobě, své heslo a případně další nepovinné informace, jako přezdívku nebo email. Kontrola, zda se nově registrovaný uživatel již v databázi nenalézá, zde není vhodná. Při úspěchu přijde uživateli zpětná vazba, obsahující jeho uživatelské id, kterým se bude v kombinaci se zvoleným heslem v aplikaci nadále prokazovat.

3.2.7 Detail uživatele

Detail uživatele lze získat ve dvou verzích. První verze je jako neoprávněný uživatel, kdy informace obsažené v odpovědi nejsou citlivé, např. jméno, přezdívka nebo fotka. Druhá verze je jako oprávněný uživatel, např. když si uživatel prohlíží svůj vlastní profil, kdy jsou zobrazeny i citlivější údaje, jako email nebo postup ve hře při hodnocení pokrmů.

V první verzi je odpověď sestavena pouze ze záznamu z tabulky uživatelů, ve druhé je možné přiložit informace i z tabulek hry, hodnocení i odložených hodnocení. Informace jako heslo, nebo role v systému se nesmí zobrazovat nikdy nikomu. Pokud uživatel heslo zapomene, musí si nastavit nové. Tento přístup je v současnosti hojně využívaný a přináší vysokou úroveň zabezpečení.

3.2.8 Vztahy mezi uživateli

Vztahy může vytvářet pouze přihlášený uživatel, a to vždy pouze způsobem, kdy přihlášený uživatel začne následovat jiného uživatele. Opačným směrem vytváření vazby nemá smysl a ani se nepoužívá. Vždy může existovat nejvýše jeden záznam o vztahu kdy uživatel A následuje uživatele B. Více takových záznamů by bylo přebytečných a mohly by způsobit potíže. Pro tyto dva uživatele již může existovat jen jediný záznam, kdy uživatel B následuje uživatele A.

K prohlížení těchto vztahů je vždy nutno upřesnit výchozího uživatele a poté zvolit směr vztahu, tedy zda se má zobrazit seznam uživatelů následovaných tímto uživatelem, nebo seznam uživatelů, kteří následují tohoto uživatele.

3.2.9 Přidání pokrmu

Přidávat pokrmy smí pouze přihlášený uživatel. V požadavku je nutné zaslat citlivé údaje, proto musí být použita metoda POST. Ten přináší výhodu neomezené velikosti parametrů. Protože je API implementováno pomocí architektury REST, požadavek by měl obsahovat všechny informace potřebné k vykonání operace, tedy přihlašovací údaje, název pokrmu a identifikátor stravovacího zařízení.

Aplikační rozhraní, které přijalo tento požadavek, musí ověřit existenci zmíněného zařízení. Pokud existuje, získá si o něm informace z databáze Foursquare, ty uloží do vlastní databáze a teprve poté přidá nový pokrm. Při tomto procesu se může leccos pokazit a uživatel by o tom měl být informován, proto je vhodné zaslat zpětnou vazbu o úspěchu či

neúspěchu operace. Při úspěchu se k odpovědi připojí i identifikátor nově přidaného pokrmu, pro možnost následného přidání hodnocení. Za takovýto přínos do databáze musí být následně odměněn body odpovídající herní kategorie a také zkušenostními body.

3.2.10 Přidání hodnocení pokrmu

Přidávat hodnocení k pokrmům může opět pouze přihlášený uživatel. Navíc je potřeba znát identifikátor pokrmu, který uživatel získal buď z odpovědi o úspěšném přidání nového pokrmu, nebo z detailu pokrmu. Povinnými parametry jsou dále hodnocení kvality, poměru ceny ku kvalitě, velikosti a prezentace. Volitelnými parametry jsou cena, měna a komentář.

Každý záznam v tabulce hodnocení je jednoznačně identifikován pomocí identifikátoru uživatele a času vytvoření záznamu. Tyto dva údaje k jednoznačné identifikaci záznamu stačí, za předpokladu že jsou požadavky zasílány z mobilní aplikace uživatelem. Je potřeba kontrolovat unikátnost každého nového záznamu před jeho vložením do databáze pro případ, že by se někdo snažil vkládat hodnocení rychle za sebou, například pomocí skriptu. Při tomto procesu se může leccos pokazit, proto je vhodné zaslat uživateli zpětnou vazbu o úspěchu, či neúspěchu operace. Za takovýto přínos do databáze musí být následně odměněn body pro postup ve hře a také zkušenostmi.

Kapitola 4

Implementace

Kapitola implementace se zabývá nejdůležitějšími body, poznatky a problémy, které se v průběhu realizace vyskytly. Nastiňuje jejich různá řešení a detailněji popisuje ta zvolená. Projekt Yumified není pouze projektem pro bakalářskou práci, ale jeho vývoj bude pokračovat, proto by v rámci této práce neměl být považován za hotový. Existuje ještě spousta návrhů a funkcí, které nebyly implementovány, ale s jejich realizací se počítá do budoucna.

4.1 Implementace databáze

Pro implementaci databáze byl zvolen dialekt SQL jazyka, MySQL. Byl vybrán především pro svou celosvětovou rozšířenost, popularitu, obrovskou podporu a kvalitní dokumentaci. Struktura databáze je popsána jediným SQL souborem `create_all.sql`, který je k nalezení na přiloženém CD. Databáze sestává z deseti tabulek, z čehož tři tabulky jsou vazební. Následující odstavce se zabývají problémy a netypickými řešeními, které se při tvorbě databáze vyskytly.

4.1.1 Primární klíč vytvořený konkatenací cizího klíče a čísla

V databázi se nachází pouze dvě tabulky, které mají naprosto vlastní primární klíč. Jednou z nich je tabulka `User`, kde je primární klíč (`userid`) tvořen číslem a zvyšován je pomocí `auto-increment`. Druhá tabulka je `Facility`, jejíž primární klíč (`facid`) je tvořen textovým řetězcem o délce 24 znaků. Tento klíč není nijak vytvářen, ale je přebírán z databáze Foursquare, kde je pojmenován jako `venue_id`. Tabulka `Dish` má sice taktéž svůj vlastní primární klíč, nemusí již ale být naprosto unikátní.

Možná řešení primárního klíče tabulky `Dish`

Primární klíč pro tabulku `Dish` sebou nese mnoho omezení. Tento klíč je v databázi často používán jako cizí klíč, proto je vhodné, aby práce s ním byla co nejjednodušší a nejpohodlnější. Zároveň musí jednoznačně určovat každý záznam této tabulky.

Je možné vyřešit tento problém například zavedením primárního klíče pokrmu jako čísla, zvyšujícího se pomocí `auto-increment` a toto číslo nechat společné pro všechny pokrmy všech zařízení. To by však vyžadovalo datový typ o vysokém rozsahu, nejspíše `bigint`, který ale přináší nevýhody pomalejší manipulace a nepřehlednosti pro člověka, vývojáře i uživatele.

Dalším možným řešením by mohlo být vytvoření složeného primárního klíče z cizích klíčů z tabulek `Facility` a `User`. Použit by byl primární klíč toho uživatele, který pokrm do databáze přidal. Nevýhodou je, že takovýto záznam stále není jednoznačně identifikovatelný, neboť stejný uživatel může pro jeden podnik přidat více pokrmů. Proto potřebuje další odlišující údaj, např. čas přidání nebo pořadové číslo. S takovýmto složeným primárním klíčem se ale nepohodlně zachází a náročnost implementace roste. Nakonec také nemusí být vhodné spojovat uživatele přímo s pokrmem, neboť v budoucnu může uživatel svůj účet zrušit, což by mohlo způsobit problémy.

Konečné řešení primárního klíče tabulky `Dish`

Primární klíč tabulky `Dish` je vytvářen konkatenací primárního klíče tabulky `Facility` a pořadového čísla, v rámci daného stravovacího zařízení. Velikost pořadového čísla je omezena v rozsahu 11111 – 99999, což poskytuje 88888 možných záznamů pokrmů pro jedno zařízení. Primární klíč tabulky `Dish` pak může vypadat například: `502cf17ce4b0260c90a8d0e011113`, kde posledních pět čísel určuje pořadové číslo pokrmu. Tento způsob přináší jednoduchost při používání, neboť se jedná o jediný údaj, a zároveň dostatečnou a logickou identifikaci pro každý pokrm.

4.1.2 Primární klíč složený z cizího klíče a hodnoty

Pokud má každá tabulka svůj vlastní primární klíč, je práce s takovou databází pohodlná. Pokud však má tabulka obsahovat obrovské množství záznamů, některé datové typy nemusí stačit, a poté ty dostatečně velké, např. `bigint` mohou způsobit pomalejší procházení a vyhledávání. V takových případech je vhodné použít pro jednoznačnou identifikaci záznamů cizí klíče. Ve vyvíjené databázi tento způsob využívá 7 z 10 tabulek.

Tabulka `Progress`

Tabulka `Progress` pro identifikaci svých záznamů využívá primární klíč tabulky `Facility`, k němuž přidává čas vytvoření záznamu a náhodné číslo. Toto číslo je ke klíči přidáno pro případ, kdy by v jeden okamžik dva uživatelé hodnotili pokrm z totožného stravovacího zařízení. Při běžném používání je taková situace velmi nepravděpodobná, ale může to také být záměr dvou domluvených uživatelů. V takovém případě se nabízí dvě možnosti, buď přidání záznamu odmítnout, nebo připojit již zmíněné náhodně vygenerované číslo o dostatečně velkém rozsahu. Možnost připojit k záznamu místo náhodného čísla primární klíč uživatele, který vyvolal změnu, byla zamítnuta, protože není vhodné spojovat záznam s uživatelem z důvodů zmíněných v části 4.1.1. Výsledný klíč je tedy deklarován následovně:

```
PRIMARY KEY (‘facid‘, ‘progtime‘, ‘progkey‘),
FOREIGN KEY (‘facid‘) REFERENCES facility(‘facid‘) ON DELETE CASCADE
```

Tabulky `Rated`, `Unrated`, `Photo` a `Likes`

Tabulky `Rated`, `Unrated`, `Photo` a `Likes` využívají k jednoznačné identifikaci každého záznamu primární klíč z tabulky `User` v kombinaci s časem vytvoření záznamu. Při běžném používání je tato dvojice dostačující. Pokud by se uživatel pokoušel vložit více záznamů rychle za sebou, například skriptem, nejedná se o běžné použití a v takovém případě by mohly být další záznamy odmítnuty. Tento problém je ale záležitost aplikačního rozhraní. Implementace složeného primárního klíče je demonstrována následujícím příkladem:

```
PRIMARY KEY (‘userid‘, ‘ratedtime‘),
FOREIGN KEY (‘userid‘) REFERENCES user(‘userid‘) ON DELETE CASCADE
```

Tabulka `Game`

Tabulka `Game` obsahuje taktéž údaj o čase vytvoření záznamu, nepotřebuje jej ale k jednoznačné identifikaci záznamu. K tomu slouží identifikátor kategorie stravovacího zařízení v kombinaci s primárním klíčem uživatele. Údaj o čase je zde připojen ke složenému primárnímu klíči spíše dodatečně. Lépe je vše vidět na následujícím příkladu:

```
PRIMARY KEY (‘userid‘, ‘gamecatid‘, ‘gametime‘),
FOREIGN KEY (‘userid‘) REFERENCES user(‘userid‘) ON DELETE CASCADE
```

4.1.3 Vazební tabulka `Follow`

Zvláštností ve vyvíjené databázi je vazební tabulka `Follow`, jejíž primární klíč je složen ze dvou cizích klíčů, přičemž oba jsou primární klíče z tabulky `User`. V databázi je povolena vždy jen jedna variace těchto klíčů, přesto je k nim přidán ještě čas vytvoření záznamu. Implementace je znázorněna následujícím příkladem:

```
PRIMARY KEY (‘followeruserid‘, ‘followeduserid‘, ‘followtime‘),
FOREIGN KEY (‘followeruserid‘) REFERENCES user(‘userid‘) ON DELETE CASCADE,
FOREIGN KEY (‘followeduserid‘) REFERENCES user(‘userid‘) ON DELETE CASCADE
```

4.2 Implementace aplikačního rozhraní

Pro pohodlné používání databáze byly navrženy koncové body, v praxi nazývané jako endpointy, které jsou popsány tabulkou 4.1. V následujících kapitolách jsou popisovány některé problémy a řešení z implementace těchto endpointů.

4.2.1 Implementace webových stránek

Tvorba webového aplikačního rozhraní není primárním účelem PHP frameworku Symfony. Ten je zaměřen na tvorbu moderních, dynamických a bezpečných webových stránek a vytváření aplikačního rozhraní je spíše druhotná záležitost. Přesto byla pro projekt Yumified vytvořena jen úvodní stránka s logem a dvěma, zatím nefunkčními, obrázkovými odkazy ke stažení aplikace pro Android a iOS. Další plánované stránky jsou především dokumentace aplikačního rozhraní a administrátorská sekce.

Pomocí webového prohlížeče lze zasílat všechny GET požadavky, jež jsou uvedeny v tabulce 4.1 a získávat tak jejich výsledky i bez použití mobilní aplikace nebo terminálového nástroje. Tyto stránky jsou automaticky generovány z navracených výsledků pomocí serializéru. Ten je dostatečně inteligentní a dokáže z asociativních polí na vstupu vytvořit čitelnou stránku, zobrazitelnou ve webovém prohlížeči. Další výhodou je možnost volby podoby výsledku. Ten může být buď ve formě XML dokumentu, nebo pomocí zápisu v JSON. Další formáty je možné přidat jednoduchým nastavením serializéru, pro běžné potřeby ale tyto dva stačí.

4.2.2 Vytvoření tříd entit

Symfony framework využívá pro manipulaci se záznamy v databázi princip objektově relačního mapování. To přináší možnost manipulovat s jednotlivými záznamy tabulek jako s běžnými objekty. Knihovna Doctrine, která je implicitní součástí Symfony frameworku, nabízí funkci automatického generování tříd pro definici těchto objektů. Jedná se sice jen o vytvoření atributů a základních metod `set` a `get` pro každý atribut, ale vývojáři to zajisté přinese velkou úsporu času, obzvláště u velkých databází se spoustou tabulek. Vygenerované třídy je možné dále upravovat a přidávat vlastní metody.

typ	**endpoint**	**povinné parametry**						
GET	facilities	lat	lng					
GET	facility	id						
GET	menu	id						
GET	dish	id						
GET	top_around	lat	lng					
GET	progress	facid						
GET	userinfo	userid						
GET	comment	id						
GET	followed	userid						
GET	following	userid						
POST	regusers	name	surname	password				
POST	connects	userid	password					
POST	editusers	userid	password					
POST	follows	userid	password	followeduserid				
POST	adddishes	userid	password	facid	name			
POST	addratings	userid	password	dishid	price	quality	presentation	size
POST	addunratings	userid	password	dishid				
POST	addphotos	userid	password	dishid	url			
POST	addlikes	userid	password	photouserid	phototime			
POST	games	userid	password					

Tabulka 4.1: Tabulka endpointů a povinných parametrů aplikačního rozhraní

4.2.3 Ověření korektnosti parametrů

Implementované aplikační rozhraní přijímá požadavky dvou metod, GET a POST. U požadavků typu GET jsou parametry předávány jako součást URL a jsou tedy čitelné i pro uživatele. U požadavků typu POST jsou parametry součástí těla dotazu, nejsou běžně čitelné pro uživatele a také se neukládají do cache a historie prohlížeče. Z tohoto důvodu jsou požadavky POST vhodné pro zasílání citlivých údajů, např. jména a hesla.

Symfony framework přináší knihovny pro automatické zachycení a parsování obou druhů požadavků. Parametry jsou pak vývojáři dostupné ve formě asociativního pole, se kterým se pracuje podstatně lépe.

Korektnost přijatých parametrů u metody GET je možné ověřit definováním datového typu a rozsahu pomocí regulárního výrazu, zapsaného v anotačním komentáři. Parametry, které jsou v URL navíc, jsou automaticky ignorovány a nepředstavují pro aplikační rozhraní hrozbu.

Dalším stupněm ověření, společným pro GET i POST, je kontrola neprázdnosti povinných parametrů, např. kontrola prázdného textového řetězce (""). Důležité je také ověřit rozsah zadané zeměpisné šířky a délky (`lat` a `lng`). Tyto hodnoty se mohou pohybovat pouze v rozmezí: lat <-90; 90> a lng <-180; 180>.

Poslední stupeň ověření se týká primárního klíče záznamu, zadaného jako parametr požadavku. Před provedením požadované operace, např. přidání hodnocení pro daný pokrm, je ověřena existence zadaného pokrmu v databázi. Pokud záznam není nalezen, provedení požadavku je zamítnuto. Ověření existence a správného přihlášení uživatele demonstruje následující příklad:

```
$user = $repository->findOneBy(array("userid" => $request->
                                                    request->get("userid")));
if (($user == null) or
    ($user->getUserpassword() != $request->request->get("password")))
      return array("code" => "fail");
```

4.2.4 Zisk informací o stravovacích zařízení

Na počátku svého fungování, databáze neobsahuje žádné záznamy o stravovacích zařízeních. Pokud si uživatel vyžádá seznam nejbližších zařízení, aplikační rozhraní jeho požadavek přepošle na Foursquare API. Odpověď od Foursquare přepracuje do vlastní podoby a zašle ji uživateli. Teprve ve chvíli, kdy uživatel přidává do databáze pokrm, jsou informace o stravovacím zařízení ukládány do vlastní databáze.

Při vyhledávání nejbližších stravovacích zařízení lze upravovat přesnost výsledků zadáním velikosti okruhu pro hledání. Při vynechání tohoto parametru se může stát, že se ve výsledcích neobjeví to zařízení, ve kterém se právě uživatel nalézá. To je nepřípustný nedostatek, proto je potřeba tento parametr zadávat vždy. Nabízí se tři způsoby získávání nejbližších zařízení.

V prvním způsobu by byl na Foursquare zaslán jen jeden dotaz s malým parametrem `radius`, např. 1000 metrů. V tomto případě jsou získané výsledky velmi přesné a pokud se uživatel nalézá v některém stravovacím zařízení, informace o něm budou vždy na prvním místě. Díky zaslání jen jednoho dotazu na Foursquare je tento způsob nejrychlejší. Nevýhodou ale může být malý počet záznamů o jednotlivých zařízeních. Tento nejjednodušší způsob je demonstrován následujícím příkladem:

```
$client = $this->container->get("jcroll_foursquare_client");
$client->addToken($oauthToken);
$gvc= $client->
       getCommand("venues/search",
                  array("ll" => $ll,
                        "radius" => "1000",
                        "categoryId" => "4d4b7105d754a06374d81259"));
$response = $gvc->execute();
```

V druhém způsobu by docházelo k cyklickému zasílání dotazů na Foursquare, pokaždé s větším parametrem `radius`. Odpovědi všech dotazů by se postupně slučovaly a duplicitní záznamy se zahodily. Tento způsob přináší detailní seznam stravovacích zařízení, z blízkého i vzdálenějšího okolí, a výsledků může být mnoho. Nevýhodu je velmi dlouhá doba provádění. Časový rozdíl mezi prvním a druhým způsobem se pohybuje v řádu sekund.

Jako optimální způsob se ukázalo zaslání tří dotazů, s poloměry 1000 metrů, 2000 metrů a poslední bez poloměru. Odpovědi těchto tří dotazů se opět sloučí, duplicitní záznamy se zahodí a výsledkem je dostatečně detailní pole záznamů o blízkých i vzdálených zařízeních. Doba provádění se zde pohybuje v řádu desetin sekundy, což je při zisku takto detailního výsledku přijatelné.

4.2.5 Stránkování odpovědi

V současných aplikacích je trendem nezasílat ihned úplně všechny výsledky, ale zaslat jich jen předem určený počet, a na požádání zaslat další. Tomuto principu se říká stránkování a slouží k urychlení komunikace, díky menšímu objemu dat přenášeného po síti. Projekt Yumified tento princip využívá u endpointů `facilities`, `menu`, `comment`, `top_around`, `followed` a `following`.

Jeho realizace je založena na zasílání dvou parametrů, `page` a `page_limit`, které specifikují číslo strany a počet záznamů na jedné straně. Z těchto dvou čísel je vypočten počáteční a koncový index, určující které záznamy z pole výsledků budou předány uživateli.

S tímto se pojí také položka `next_page`, nacházející se v odpovědi zmíněných endpointů. Její hodnotou je URL, na které lze získat další stranu. Pokud ale již uživatel obdržel poslední stranu, je potřeba aby ta položku `next_page` neobsahovala. Toho lze docílit jednoduchým ověřením, zda se koncový index rovná délce pole výsledků zmenšené o jedna. V takovém případě se jednoduše položka `next_page` nepřipojí do odpovědi. Toto téma je shrnuto a demonstrováno následujícím příkladem pro pole pokrmů:

```
$dishesCount = count($dishes);
$start = ($params[page] - 1) * $params[page_limit];
$stop = ($params[page] * $params[page_limit]) - 1;

if ($start > $dishesCount-1)
    return array("count" => 0, "menu" => null, "next_page" => null);
if ($stop > $dishesCount-1)
    $stop = $dishesCount-1;

$dishes = $this->getSubarray($dishes, $start, $stop);
if ($dishesCount-1 == $stop)
    $next_page = null;
```

4.2.6 Zamezení zobrazení citlivých údajů

Některé informace o uživateli, například jeho heslo, by se nikdy neměly dostat mimo aplikační rozhraní. Uvnitř něj je potřeba s heslem pracovat, především při ověřování uživatele, ale nikdy se nesmí dostat do odpovědi. Primární zabezpečení spočívá v tom, manuálně takové informace do odpovědi nepřidávat. Jsou ale situace, kdy se díky propojenosti tabulek v databázi, například k záznamům o hodnocení jídel, automaticky přidají informace o hodnotitelích. Tento únik vývojář nemusí očekávat a je potřeba jej bezpečně ošetřit.

Toho lze docílit správným nastavením serializéru. Každá odpověď, vytvořená v aplikačním rozhraní, musí projít skrze serializér, který ji převede do požadované podoby (JSON nebo XML). Zde je možné nastavit, které položky mohou být do odpovědi přidány a které ne. Serializér tak lze využít jako poslední zabezpečovací bariéru před úniky citlivých informací.

4.2.7 Přidání a hodnocení pokrmu

Přidání pokrmu a přidání hodnocení je rozděleno do dvou samostatných požadavků, které na sebe navazují. Pokud se uživatel pokusí přidat hodnocení k pokrmu, který v databázi neexistuje, je mu navrácena odpověď o chybě.

Přidání pokrmu

Pokud přidává nový pokrm, musí mezi parametry zadat své přihlašovací údaje, identifikátor stravovacího zařízení a název nového pokrmu. Nejprve se ověří, zda zadané stravovací zařízení existuje v databázi. Pokud ne, je zaslán dotaz na Foursquare. Pokud ani Foursquare toto zařízení ve své databázi nenalezne, uživateli je navrácena chybová odpověď. Pokud jej nalezne, do vlastní databáze se přidá záznam s novým stravovacím zařízením a následuje přidání nového pokrmu. Za takové přispění do databáze uživatel získá zkušenostní body. V odpovědi o úspěšném přidání pokrmu do databáze je obsažen i identifikátor nově přidaného pokrmu.

Přidání hodnocení pokrmu

Pokud chce uživatel vzápětí přidat i jeho hodnocení, využije tento identifikátor jako parametr. Při zaslání požadavku na přidání hodnocení se musí ověřit existence zadaného pokrmu. Dále se ověřuje, zda tento uživatel nemá pro tento pokrm v databázi uložen záznam o odloženém hodnocení. Pokud ano, je smazán a vzápětí je vytvořen nový záznam o hodnocení pokrmu.

Aktualizace průměrného hodnocení pokrmu

Při úspěšném přidání nového hodnocení musí dojít hned k několika operacím. První operací je aktualizace průměrného hodnocení právě ohodnoceného pokrmu. Znamená to nalézt všechny záznamy o hodnocení daného pokrmu v databázi, spočítat jejich nový průměr a ten uložit do položky `dishrating` v záznamu pokrmu.

Aktualizace průměrného hodnocení stravovacího zařízení

Další operace zahrnuje zprůměrování hodnot položek `dishrating` ze všech pokrmů ze stejného stravovacího zařízení. Nový průměr je uložen do položky `facrating` v záznamu příslušného stravovacího zařízení.

Přidání záznamu do tabulky `Progress`

Předposlední operace spočívá ve vytvoření nového záznamu do tabulky `Progress`, kam je uložena nová hodnota průměrného hodnocení daného stravovacího zařízení a čas vytvoření záznamu. Tato data později slouží k vykreslení grafu v mobilní aplikaci.

Přidání nebo aktualizace záznamu v tabulce `Game`

Poslední operace je vytvoření, nebo aktualizace záznamu o sběru bodů v tabulce `Game`. Za každé přidané hodnocení pokrmu je uživatel odměněn body, které zvedají prestiž a důvěryhodnost jeho hodnocení. Jedná se o jiné body, než jaké může získat za přidání pokrmu nebo fotografie. Body jsou navíc děleny do kategorií podle druhu stravovacího zařízení, např. italská kuchyně, čínská kuchyně, indická kuchyně, atd. Pokud tedy uživatel hodnotil pokrm z čínské kuchyně a doposud nemá záznam z této kategorie, je nutné vytvořit nový záznam. V opačném případě dojde pouze k přidání bodů do tohoto záznamu.

4.2.8 Vytvoření primárního klíče pokrmu

Z důvodů zmíněných v kapitole 4.1.1 je nutné vytvářet primární klíč pro každý nově přidaný pokrm. Jeho tvorba spočívá v konkatenaci identifikátoru stravovacího zařízení s pořadovým číslem. Metoda `createNewDishId` nejprve všechny záznamy pokrmů pro dané zařízení seskládá do pole a zjistí jeho velikost. Pokud je nulová, jedná se o první pokrm a číslo bude rovno 11111. Jinak je pole pokrmů seřazeno vzestupně podle položky `dishid`. Pořadové číslo nového pokrmu je zadáno o jedna větší než je pořadové číslo posledního záznamu v poli pokrmů.

4.2.9 Kontrolér pro správu databáze

Je velice pravděpodobné, že při ostrém provozu bude docházet k vytváření spousty duplicit, falešných hodnocení a neaktivních profilů uživatelů. Proto je vhodné před nasazením do provozu vytvořit nástroj pro správu databáze. Projekt Yumified má v základní verzi k dispozici jen pár jednoduchých metod pro vypsání položek v jednotlivých tabulkách a mazání profilu uživatelů. Je zapotřebí dalších nástrojů pro vyhledávání a mazání bezcenných záznamů, aby byla databáze použitelná pro běžný provoz.

Kapitola 5

Testování

Testování aplikačního rozhraní a databáze bylo prováděno ve třech fázích: při implementaci pomocí nástroje HTTPie[1] v terminálu, po implementaci pomocí skriptů a po nahrání na internet pomocí mobilní aplikace.

5.1 Testování pomocí HTTPie

Terminálový nástroj HTTPie umožňuje jednoduše a rychle zasílat požadavky GET a POST. U požadavků GET je nutné escapovat všechny ampersandy "&" při zadávání URL s parametry. U požadavků POST jsou parametry zadávány za URL odděleny mezerou. Výhodou tohoto nástroje je přehledné strukturování odpovědi a barevné odlišení klíčů a hodnot.

Tímto nástrojem byly prováděny testy funkčnosti všech metod bezprostředně při jejich implementaci a byly tak odhaleny prvotní chyby a nedostatky. Následuje příklad zadání požadavků GET a POST v nástroji HTTPie:

```
# GET
http http://localhost/web/facilities.json?lat=49\&lng=0\&page=2\&page_limit=8

# POST
http http://localhost/web/adddishes.json userid=111112 password=secret \
                                         facid=502cf17ce4b0260c90a8d0e0 \
                                         name="Hovězí steak"
```

5.2 Testování pomocí skriptu

Po implementování stanovených metod následovalo testování pomocí skriptů. Pro každý požadavek (`facilities`, `menu`, `adddishes`, atd.) byl vždy vytvořen jeden skript pro jeho testování. Skripty se zaměřují především na chybné vstupní parametry, ať už rozsahem nebo datovým typem. Implementovány byly jako množina příkazů pro HTTPie sestavených za sebou.

Při tomto testování byl odhalen problém, způsobený zadáním neexistujícího identifikátoru stravovacího zařízení. Tento identifikátor je aplikačním rozhraním přeposílán na Foursquare, který jej nepozná a již samotné provedení příkazu odeslání požadavku způsobuje chybu. Tuto chybu se nepodařilo odchytit ani uzavřením příkazu do bloku `try{...}catch()`.

[1] https://github.com/jakubroztocil/httpie

Řešením by mohlo být kontrola validního identifikátoru již v aplikačním rozhraní, k čemuž je ale potřeba znát správný hashovací algoritmus.

5.3 Testování pomocí mobilní aplikace

V průběhu vývoje, ale hlavně v jeho závěru, bylo aplikační rozhraní testováno pomocí mobilní aplikace. Primární použití aplikačního rozhraní je zamýšleno právě pro vyvíjenou aplikaci, proto se tímto testováním strávilo nejvíce času.

Aplikaci testovalo deset osob, osm zkušených uživatelů mobilních aplikací a dva méně zkušení. Každý uživatel měl zadány úkoly k provedení a poté mu byl dán neomezený čas pro libovolnou manipulaci. V průběhu celého závěrečného testování nebyla odhalena chyba aplikačního rozhraní ani databáze. Výhodou použití aplikačního rozhraní v kombinaci s mobilní aplikací spočívá v tom, že sama aplikace ověřuje správnost zasílaných parametrů, což minimalizuje riziko vzniku chyby.

Kapitola 6

Závěr

Cílem této práce bylo vytvořit databázi pro ukládání pokrmů, hodnocení pokrmů, stravovacích zařízení, které je servírují, a uživatelů, kteří je hodnotí. K těmto nejdůležitějším informacím přibyly další, umožňující doplňkové funkce. Databáze byla popsána pomocí MySQL a některé implementační detaily byly rozebrány v částech návrhu a implementace databáze, 3.1 a 4.1.

Aby se s databází dobře pracovalo, dalším úkolem byla implementace webového aplikačního rozhraní. Podmínkou v zadání práce byla realizace pomocí PHP frameworku libovolného výběru. Pro svou celosvětovou popularitu, rozšířenost a kvalitní dokumentaci byl vybrán framework Symfony 2. Množství usnadnění a bezpečnostních opatření přinesly jeho knihovny pro zpracování parametrů z GET i POST požadavků a serializér výstupních dat. Některé implementační detaily byly popsány v částech 3.2 a 4.2.

Implementované endpointy byly popsány tabulkou 4.1. Bylo využito jak požadavků metody GET, tak požadavků metody POST. Aplikační rozhraní dokáže informace ukládat i poskytovat, přičemž ukládat mohou pouze přihlášení uživatelé. Většina požadavků pro získání dat je proveditelná pomocí GET, proto je možné je získávat i přes webový prohlížeč.

Testování proběhlo ve třech fázích, pomocí terminálového nástroje HTTPie, pomocí sady skriptů a pomocí mobilní aplikace. Většina zjištěných problémů byla opravena. Ty, které se opravit nepodařilo, vyžadují hlubší zkoumání příčiny a řešení, ale i tak nezpůsobují nekonzistenci nebo jiné problémy v databázi.

Výsledné řešení bylo nahráno na bezplatný webhosting a je dostupné k odzkoušení na URL: `http://www.apitest.g6.cz/web/`. Na této URL je k nalezení pouze úvodní obrazovka. Pro získání např. seznamu blízkých stravovacích zařízení se za zmíněné URL připojuje název endpointu, formát odpovědi a parametry. Výsledná URL pak vypadá například:

`http://www.apitest.g6.cz/web/facilities.json?lat=49&lng=0&page=1`

Literatura

[1] WWW STRÁNKY. *PHP PDO Introduction* [online]. 2015 [cit. 2014-12-21]. Dostupné na: <http://php.net/manual/en/intro.pdo.php>.

[2] TICHÝ, J. *Doctrine 2: Query Builder a nativní SQL* [online]. 2010-11-18 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/doctrine-2-query-builder-a-nativni-sql/>.

[3] VRÁNA, J. *NotORM PHP library* [online]. 2010 [cit. 2014-12-21]. Dostupné na: <http://www.notorm.com/#why>.

[4] VRÁNA, J. *Databáze v PHP elegantně s NotORM* [online]. 2010-05-06 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/databaze-v-php-elegantne-s-notorm/>.

[5] WIKIPEDIE: OTEVŘENÁ ENCYKLOPEDIE. *Objektově relační mapování* [online]. 2014-05-07 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD&oldid=11440457>.

[6] TRÖSTER, F. *ORM frameworky pro PHP5* [online]. 2010-01-27 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/orm-frameworky-pro-php5-obecny-uvod/>.

[7] TICHÝ, J. *Doctrine 2: DQL* [online]. 2010-11-03 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/doctrine-2-dql/>.

[8] WIKIPEDIE: OTEVŘENÁ ENCYKLOPEDIE. *Webová služba* [online]. 2014-09-26 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Webov%C3%A1_slu%C5%BEba&oldid=11883878>.

[9] WIKIPEDIE: OTEVŘENÁ ENCYKLOPEDIE. *SOAP* [online]. 2013-11-29 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=SOAP&oldid=12328885>.

[10] WIKIPEDIE: OTEVŘENÁ ENCYKLOPEDIE. *Distributed Component Object Model* [online]. 2014-12-19 [cit. 2014-12-21]. Dostupné na: <http://en.wikipedia.org/w/index.php?title=Distributed_Component_Object_Model&oldid=638686939>.

[11] KOSEK, J. *PHP a XML*. [b.m.]: Grada Publishing, 2009. ISBN 978-80-247-1116-4.

[12] Wikipedie: Otevřená encyklopedie. *Web Services Description Language* [online]. 2013-06-22 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Web_Services_Description_Language&oldid=12112137>.

[13] Wikipedie: Otevřená encyklopedie. *Remote procedure call* [online]. 2014-06-04 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Remote_procedure_call&oldid=11535913>.

[14] Wikipedie: Otevřená encyklopedie. *Representational State Transfer* [online]. 2014-10-20 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=12273280>.

[15] Malý, M. *REST: architektura pro webové API* [online]. 2009-08-03 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>.

[16] WWW stránky. *Twilio: REST API* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <http://www.twilio.com/docs/api/rest/applications>.

[17] Hanák, D. *Stopařův průvodce REST API* [online]. 2013-11-17 [cit. 2014-12-21]. Dostupné na: <http://www.itnetwork.cz/stoparuv-pruvodce-rest-api>.

[18] Fischer, L. *A Beginner's Guide to HTTP and REST* [online]. 2013-01-09 [cit. 2014-12-21]. Dostupné na: <http://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest-net-16340>.

[19] WWW stránky. *Twitter: REST APIs - GET* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <https://dev.twitter.com/rest/reference/get/statuses/user_timeline>.

[20] Wikipedie: Otevřená encyklopedie. *JavaScript Object Notation* [online]. 2014-12-12 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=12278357>.

[21] WWW stránky. *Twitter: REST APIs* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <https://dev.twitter.com/rest/public>.

[22] WWW stránky. *Google+ API* [online]. 2014-05-03 [cit. 2014-12-21]. Dostupné na: <https://developers.google.com/+/api/>.

[23] WWW stránky. *The Foursquare API* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <https://developer.foursquare.com/overview/>.

[24] WWW stránky. *Graph API Reference* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <https://developers.facebook.com/docs/graph-api/reference>.

[25] Wikipedie: Otevřená encyklopedie. *Framework* [online]. 2014-08-26 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/w/index.php?title=Framework&oldid=11780814>.

[26] Wikipedie: Otevřená encyklopedie. *Model–view–controller* [online]. 2014-12-12 [cit. 2014-12-21]. Dostupné na: <http://cs.wikipedia.org/wiki/Model-view-controller>.

[27] Skvorc, B. *Best PHP Frameworks for 2014* [online]. 2013-12-28 [cit. 2014-12-21]. Dostupné na: <http://www.sitepoint.com/best-php-frameworks-2014/>.

[28] WWW stránky. *Comparing Laravel, CodeIgniter, and CakePHP* [online]. 2012-08-02 [cit. 2014-12-21]. Dostupné na: <https://nerdmom.wordpress.com/2012/08/02/comparing-laravel-codeigniter-cakephp/>.

[29] Gutmans, A., Bakken, S. S. a Rethans, D. *Mistrovství v PHP5.* [b.m.]: Computer press, 2005. ISBN 80-251-0799-X.

[30] WWW stránky. *Phalcon Documentation: About* [online]. 2014 [cit. 2014-12-21].

[31] WWW stránky. *Symfony at a Glance* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <http://symfony.com/at-a-glance>.

[32] Tröster, F. *ORM frameworky pro PHP5: Doctrine ORM* [online]. 2010-02-03 [cit. 2014-12-21]. Dostupné na: <http://www.zdrojak.cz/clanky/orm-frameworky-pro-php5-doctrine-orm/>.

[33] WWW stránky. *CodeIgniter at a Glance* [online]. 2014-11-12 [cit. 2014-12-21]. Dostupné na: <http://www.codeigniter.com/userguide3/overview/at_a_glance.html>.

[34] WWW stránky. *Features of Yii* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <http://www.yiiframework.com/features/>.

[35] Senkevich, N. *ZF2 cheatsheet: Doctrine* [online]. 2014 [cit. 2014-12-21].

[36] WWW stránky. *Zend framework 2: About* [online]. 2014 [cit. 2014-12-21]. Dostupné na: <http://framework.zend.com/about/>.

[37] Fiori, A. *Symfony 2 vs Zend Framework 2* [online]. 2010-03-13 [cit. 2014-12-21]. Dostupné na: <http://www.andreafiori.net/symfony-vs-zend/>.

Příloha A

Obsah CD

A.1 api.zip

Tento archiv obsahuje soubory PHP frameworku Symfony 2 s rozšířením o knihovny FOSRestBundle a JcrollFoursquareApiBundle. V adresáři `src/Api/DataBundle/` se nachází soubory tohoto projektu.

- `Controller` – všechny kontroléry zajišťující fungování api,
- `Entity` – soubory s definicí všech tříd pro objektově relační mapování,
- `Resources/config/doctrine` – soubory s popisem mapování objektů na databázi,
- `Resources/config/serializer` – soubory s definicí povolených a zakázaných informací v odpovědi,
- `Resources/views/Default` – soubor s definicí titulní webové stránky.

A.2 database.zip

Archiv obsahuje soubor `create-all.sql`. Tento soubor definuje všechny tabulky, jejich atributy a datové typy.

A.3 tests.zip

Archiv obsahuje sadu skriptů, využitých pro testování funkčnosti všech metod a hlavně správných reakcí na zadání chybných parametrů.

Příloha B

Manual

B.1 Vytvoření databáze

Databázi lze vytvořit jednoduše pomocí nástroje phpMyAdmin. Po založení nové databáze stačí funkcí `import` nastavit vytvoření tabulek podle přiloženého souboru `create-all.sql`. Pro další použití je nutné znát login a heslo uživatele databáze a také název databáze.

B.2 Konfigurace aplikačního rozhraní

Archiv `api.zip` je pro lokální použití nutné rozbalit do složky `www/` nebo `www/html/`, podle konfigurace Apache každého uživatele. Pro zprovoznění na internetu je nutné archiv rozbalit do správné složky podle pokynů konkrétního webhostingu.

Po úspěšném rozbalení do správné složky je potřeba nastavit informace pro propojení s databází. Tyto údaje lze nastavit v souboru `app/config/parameters.yml`. Jedná se o údaje `database_host`, `database_port`, `database_name`, `database_user` a `database_password`. Pro lokální použití je nutné nastavit složkám `app/cache` a `app/logs` právo pro zápis.

Po úspěšném provedení tohoto kroku by aplikační rozhraní mělo být funkční, což lze nejjednodušeji ověřit zadáním následujících URL do webového prohlížeče:

```
// "hosting" je nutno nahradit za doménu, např "localhost" nebo
// "apitest.g6.cz" nebo jinou použitou
http://hosting/web
http://hosting/web/facilities.json?lat=49&lng=0
http://hosting/web/menu.json?id=502cf17ce4b0260c90a8d0e0
```

První URL by měla zobrazit úvodní stránku projektu Yumified. Druhá URL by měla zobrazit všechna stravovací zařízení na souřadnicích N 49° E 0° ve formátu JSON. Třetí URL by měla zobrazit všechny pokrmy ze stravovacího zařízení s daným identifikátorem. Protože v databázi zatím nejsou žádné pokrmy, měla by se objevit odpověď `{"count": 0}` ve formátu JSON.